Remarks

This is filed response to the final Office Action mailed December 21, 2004, rejecting the pending claims as anticipated or obvious over Real-Time Innovation, Inc.'s <u>ControlShell User's Manual</u>, Ver. 5.1, published June 1996, and Ver. 6.0, published January 1999. For the reasons below, among others, these publications do not indeed detract from patentability of the claimed invention. The Applicant therefore requests that the rejections be withdrawn.

Interview Summary

At the outset, though, the undersigned thanks Examiner Ingberg for the on-going courtesy extended in regard to this application and in particular, by way of example, in insuring that proper copies of cited references were provided with the most recently issued Office Action. With respect to the Telephone Interview conducted March 31, 2004, further thanks are extended. During that interview, Applicant's representative (the undersigned) reviewed distinctions between the prior art, including both Control Shell 5.1 and object-oriented programming in general, and the claimed invention. Those distinctions include, by way of example, that a change during configuration to a parameter of an ancestor object is effective as to a descendant object with which that parameter is associated without recompilation of those objects.

Claims Amendments

The claims are patentably distinct from the cited art *sans* amendment. To the extent relevant in this regard, if at all, the Applicant notes the Examiner's assertion and interpretation, set forth on page 4 of the action, in regard to the term "parameter."

Responding to the Examiner's concern, the Applicant is unaware of any inconsistency in its use of that term in the specification and claims. Though the Examiner suggests, on page 3 of the action, that the term is amenable to two interpretations ("inheritance of attributes and methods OR the inheritance of lines in [a third-party's] figure"), the Applicant does not necessarily adopt the Examiner's reasoning or conclusion in this regard.

The Examiner appears to conclude that Applicant uses "parameter" to refer to attributes and methods — as opposed to lines of a third-party's figure. The Applicant is unsure what the Examiner means in regard to reference to such "lines." Regardless, proper interpretation of the term requires application of law defined by Congress and the court, not mere assertion. Though,

the conclusion reached by the Examiner, at least insofar as its applying to attributes and methods, is not inconsistent with Applicant's understanding of "parameter" as used in present application.

ControlShell User's Manual, Ver. 5.1

As discussed in the Applicant's previously filed **Reply & Amendment**, ControlShell User's Manual, Ver. 5.1, published June 1996, does not teach or suggest the claimed invention. Specifically, by way of non-limiting example, there is no teaching or suggestion by that publication that, in an apparatus for configuring a control system, a change during runtime configuration to a parameter of an ancestor object is effective as to a descendant object with which that parameter is associated without recompilation of those objects.

Instead, ControlShell User's Manual, Ver. 5.1, teaches throughout that changes in objects necessitate recompilation. This is particularly evident in Chapter 5, the focus of which is a facility that generates source code files: *see,* for example, the discussion at p. 5–19. This culminates in the discussion, at Section 5.4.2, in which the authors suggest a way of minimizing the number of (re)compiles, by limiting them to files that have changed.

Although the Examiner asserts that the publication teaches applying runtime changes from the parameter of a parent object to descendant object that inherit that same parameter, support for the assertion is virtually non-existent. Specifically, in the paragraph bridging pages 32 – 33, the Examiner relies on a clause in the Manual in "... many instances without recompilation." It strains credulity to believe that this clause and the otherwise minimalist discussion of "dynamic bindings" referred to by the Examiner can be reasonably deemed to anticipate or render obvious Applicant's claimed system.

ControlShell User's Manual, Ver. 6.0

The newly cited ControlShell User's Manual, Ver. 6.0, is not prior art. The case, In re Morris Epstein, 32 F.3d 1559, 31 U.S.P.Q.2D (Fed Cir. 1994) on which the Examiner relies in asserting otherwise, is largely irrelevant for present purposes. Contrary to Epstein, the press release on which the Examiner relies in the instant application to establish a date for version 6.0 of the Control Shell product does not mention any of significant features on which the present rejection is based. Moreover, the manual for that product (to wit, ControlShell User's Manual, Ver. 6.0) does not itself disclose those features.

At the outset, the Applicant notes that the present case claims the benefit of priority of United States Patent Application No. 60/134,597, filed May 17, 1999. That application, in turn, includes disclosures dated February 3, 1999 (Integrated Design Automation — Control Strategy Configurator Architecture — DFS 5536, hereinafter "DFS 5536") and March 25, 1998 (Integrated Design Automation — Framework Classes — DFS 5537, hereinafter "DFS 5537").

The ControlShell User's Manual, Ver. 6.0, is dated January 1999. The underlying product is putatively referenced in a press release dated March 16, 1998, "RTI Announces Major New Component-Based Programming System for Building Complex ElectroMechanical Systems," PR Newswire, p316SFM064.

With respect to the manual itself, that publication pre-dates by only a few months the priority application, United States Patent Application No. 60/134,597. However, the contents of the priority application make clear that the inventors were in possession of relevant aspects of the claimed invention before the manual's publication. Thus, for example, DFS 5537, dated March 25, 1998, provides at page 33:

> Almost all objects in IDA are parameterized - i.e., their type is determined by the parameter set they support, and the data that these objects represent is contained within their associated parameters. Parameterized objects have the capability to inherit their parameter set from another Parameterized Object which acts as the definition for the new object. A Parameterized Object's definition is, itself, a Parameterized Object. Using Parameters to define an object's type, and the data associated with it, provides the following capabilities:
>
> - Parameters represent data - they aren't compiled-in behavior.
>
> - Parameterized Objects support data inheritance - a Parameterized Object inherits its structure and default values from its defining object.
>
> - Any object can override the default value of various attributes of an associated Parameter. Referred to as parameter instantiation by exception, only the Parameter attributes that differ from their defaults are instantiated, and attached to the object.
>
> - Parameters associated with a Parameterized Object can also be changed by the application of a modifier object, effectively overriding the default value(s) of any matching Parameters.
>
> - A change to a Parameter in a Parameterized Object acting as a definition is reflected in all the Parameterized Objects that are derived from the defining Parameterized Object.

- Parameterized Objects can extend their definition by adding additional Parameters.

- Parameters are organized into groups, each group containing logically-related Parameters. Groups can be defined by Foxboro, and/or defined by the user.

Further discussion of the claimed system's ability to effect change to inherited parameters during configuration without recompilation is had, for example, at pages 36 and 56 of DFS 5537. Still further discussion of these and other aspects of the claimed invention is had throughout DFS 5537 and in the historical documents referred to on page 2 of DFS 5536.

In view of the foregoing, ControlShell User's Manual, Ver. 6.0, is dated January 1999, is not prior art with respect to the instant application.

The press release "RTI Announces Major New Component-Based Programming System for Building Complex ElectroMechanical Systems" (hereinafter, the "RTI Press Release") presents facts unlike those before the court in In re Epstein, upon which the Examiner relies. In Epstein, the Federal Circuit affirmed the Patent Office's rejection of claims on account of prior art systems alleged, in abstracts published after the critical date, to have been sold prior to the critical date. Unlike the press release cited by the Examiner in the instant case, those at issue in Epstein described features of the underlying software system critical to the Patent Office rejection (as well as attributing to those systems early release dates):

> We have reviewed the abstracts and note the following about them. Each abstract contains a description of its particular software product, including the various features relied upon by the examiner in rejecting appellant's claims. Each abstract identifies the software vendor by name and provides the vendor's address and phone number. Each abstract provides information useful to potential buyers, including who to contact, price terms, documentation, warranties, training and maintenance. Each abstract states the date that the product was first released or installed, [**12] which dates range from 1977 to January 1987. Finally, all the abstracts, excepting only the abstract of Pro-Search 1.08, disclose the number of current users; these range in number from ten to fifty-eight.

In re Epstein, supra, 32 F.3d at 1565 (emphasis added).

The instant case differs. Even assuming for sake of argument that the RTI Press Release accurately describes the release date of ControlShell version 6.0, it does not — contrary to Epstein — describe features of that release germane to the claimed invention. For example, the

press release does not mention that the system permitted changes in inherited parameters to propa-
gate from ancestor objects to descendant objects during runtime configuration. Indeed, the RTI
Press Release does not even mention inheritance.

This points out the further irrelevance of In re Epstein. The court in that case pointed out
that merely admitting release date-related evidence contained in post-factor publications would not
be enough to support an on sale bar. The trier of fact must also infer that product features discloses
on in those abstracts were available at the time in question:

> Thus, because the abstracts appear on their face to be accurate and reliable, and because appel-
> lant has failed to proffer any evidence to support his arguments to the contrary, we assume the
> truthfulness of the various assertions in the abstracts. However, <u>even assuming the truthfulness of
> all the information in the abstracts, the Board's conclusions regarding "in public use or on sale" and
> evidence of skill in the art nevertheless require two additional facts to be inferred from the abstracts</u>.
> First, it must be inferred that, as of a time before the critical date, the software products, and their
> various features cited by the PTO, were not initially released or installed under a veil of secrecy. In
> other words, it must be inferred that the products and their cited features either were in "public" use,
> or were on sale for a purpose other than a bona fide experimental one. <u>Second, it must be inferred
> that the features of the software products that are relied upon by the PTO were in versions of the
> products as they existed at a time before the critical date.</u>

Id., *supra,* 32 F.3d at 1566 (emphasis added).

In reaching a conclusion that the abstracts, in Epstein, permitted such an inference, the
Federal Circuit identified several facts not present here: the relative importance of the features
mentioned in the abstracts to the products in which they purportedly appeared; that those features
not be "of the type that would be altered over the life of the product, either to upgrade the system or
to eliminate existing bugs in the system"; and that the features be "relatively broad features relied
upon by the examiner relate to the central purposes of the systems in which the features appear."
Id.

The instant case differs from Epstein. Clearly, the inheritance features could not have great
significance — or "relative importance" as the Epstein court put it — if RTI did not mention them
in their March 1998 press release. Moreover, there is little reason to believe that such features and,
specifically, those permitting changes to propagate to inherited parameters without recompilation,

would not be added to upgrade the system, e.g., in the period between March 1998 and, say, January 1999, when the ControlShell User's Manual, Ver. 6.0, was published.

Furthermore, and perhaps more significantly, there is little evidence in the manual itself that Version 6.0 of ControlShell provides an inheritance feature like that recited in the pending claims — specifically, such that a change during runtime configuration to a parameter inherited from an ancestor object is effective as to a descendant object with which that parameter is associated without recompilation of those objects.

Indeed, nowhere does ControlShell User's Manual, Ver. 6.0, recite such a capability. Though the Examiner cites the definition of "dynamic binding" at D-8 to the contrary, that definition merely provides that composite definitions are parsed at run-time:

> **dynamic binding** The composite definitions—APP, COG, FSM—for your *ControlShell* application are parsed at run-time. The run-time parsers create instances of components and connection objects and bind the connections to the component instances *dynamically* as they are encountered in the definition files.

The Examiner likewise relies, in the paragraph bridging pages 13 – 14 of the Office Action, on the discussion of run-time execution, at page 1-4 of ControlShell User's Manual. However, there's no mention of inheritance there:

1.1.4   Run-Time Execution

ControlShell's run-time executive provides the following facilities:

- ❏ At initialization time, it dynamically builds your application by *directly* loading your graphical diagrams.
- ❏ Sorts the order of execution for components in the sampled-data portion of the application based on input-output (data-flow) dependencies.
- ❏ Creates a task for each sample-rate specified by the application. Also creates tasks for the finite-state machines specified by the application.
- ❏ Full command-line menus to start, stop, and view your application as it runs.
- ❏ Lookup service to allow you or code to locate any component or signal in your application.
- ❏ Debugging facilities to connect to the *StethoScope* data-monitoring tool and to the LiveLook finite-state-machine graphical monitoring mode of *ControlShell*.

Detailed discussions about the run-time execution of *ControlShell* may be found in Chapter 14.

Furthermore, the Examiner refers to Chapter 7 of ControlShell User's Manual, Ver. 6.0, to support the contention that publication teaches or suggests that a change during runtime configuration to a parameter inherited from an ancestor object is effective as to a descendant object with which that parameter is associated without recompilation of those objects. However, nowhere is

inheritance mentioned in that Chapter — much less how changes might propagate to inherited parameters.

The same is true of Chapter 8, which is specifically directed to Component Object Group diagrams, and Chapter 9, which is directed to finite state machines. Both of these chapters delve into their respective topics at great length, yet fail to even mention inheritance or any features based thereon.

And, while the Examiner refers to Chapter 14 as "a modifier," presumably, bringing Chapters 7 – 9 together, in support of the proposition that ControlShell User's Manual, Ver. 6.0, teaches or suggests that a change during runtime configuration to a parameter inherited from an ancestor object is effective as to a descendant object with which that parameter is associated without recompilation of those objects, Chapter 14 does no such thing. As above, it does not mention inheritance nor how changes propagate with respect to inherited parameters.

Whereas the passages cited by the Examiner fail to support the view that ControlShell User's Manual, Ver. 6.0, teaches or suggests — or that the product includes a feature set such — that a change during runtime configuration to a parameter of an ancestor object is effective as to a descendant object with which that parameter is associated without recompilation of those objects, many passages of that publication suggest quite the contrary. As evident in the sections quoted below, ControlShell implemented inheritance in C++ code which required compilation for effect.

This is introduced on page 11-10, which states:

11.4 Inheritance

ControlShell supports single inheritance—in the C++ sense—for primitive components (ATC, DFC, STC) and custom data types (CT). This allows you to build upon simpler components or types to provide more complex functionality, further promoting reuse.

It is further emphasized in the paragraphs bridging page 13-1 – 13-2, providing:

13.1 Overview

Each primitive definition—CT, CM, IF, ATC, DFC, STC—in ControlShell corresponds to a C++ class. Most of the code you write will override the default behavior provided by base-class virtual functions. You may insert additional methods and data members, as needed, to aid the implementation of the components.

While almost no custom code[1] is needed for the connection types—CT, CM, IF—you must add custom code to the components to provide the functionality you need.

Inheritance ControlShell supports single inheritance—in the C++ sense—for primitive components. This allows you to build upon simpler components to provide more complex functionality, further promoting reuse. Inheritance of the method and interface connection types (CM, IF) is not supported at this time.

Auto code generation For each primitive definition, ControlShell automatically generates the C++ code stubs. You just need to add the algorithm for your components within auto-generated methods. In most cases, very little knowledge of C++ is required before you can create useful components.

And, in discussing inheritance on page 13-11, the manual suggests that inheritance has no measure of the flexibility necessary to support inheritance features like those claimed in the present application:

13.4.1.8 Inheritance

A data type (pin) and any primitive component (ATC, DFC, STC) may derive—in the C++ sense—from another one of the same type. The derived-class code may access all fields and connections of the base class directly, because the fields have been created with protected access.

Base-Class Connections When you inherit from another definition, the fields and ports defined in the base-class are imported automatically into the derived-class definition. Removing base-class ports has a very significant impact on primitive components: they will not show up as connectors in the derived-class components, effectively hiding them.

When you do hide base-class ports, the OnInstance() method of the derived-class *must* provide proper connections to these hidden ports. That is, the derived-class code *must* create pin, bubble, or interface objects (or obtain access to existing ones) and connect them to the hidden ports before calling the base-class OnInstance() method. Failing to do so may cause program crashes.

The same is true with respect to the text on the four pages cited in the index of the <u>ControlShell User's Manual</u>, Ver. 6.0, under the heading "Inheritance." On each of those pages, the manual makes clear that inherited parameters cannot be changed, only deleted. An example, is the text on page 5-38:

❑ **Inherited** This check box indicates whether or not the bubble is inherited from a base class. If so, you cannot change any of its attributes. You can only delete it from the current definition.

In view of the foregoing, the Applicants respectfully urge the Examiner to reconsider and change his position in regard to the <u>ControlShell User's Manual</u>, Versions 5.1 and Ver. 6.0, and in regard to the RTI Press Release. These publications, neither individually or together, detract from
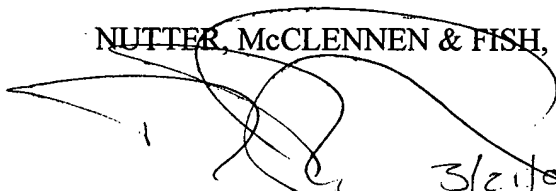
patentability of the independent claims — much less, from the claims which depend therefrom and recite further features therewith.

This filing responds in full to the final Office Action mail December 21, 2004. The prior art and other publications cited by the Examiner are addressed and shown not to detract from patentability of the claimed invention. In view thereof, the Applicant requests that the rejections be reconsidered and withdrawn.

Respectfully submitted,

NUTTER, McCLENNEN & FISH, LLP

David J. Powsner
Reg. No. 31,868

Attorney for Applicant
World Trade Center West
155 Seaport Boulevard
Boston, MA 02210-2604
Tel: (617)439-2717
Fax: (617)310-9717